Amendments to the Specification

Please amend the title to read as follows:

--Simplifying Implementation of Custom Atomic Transactions in a Programming Environment.--.

Please replace paragraph [0012] with the following amended paragraph:

[0012] FIG. 4 is a table illustrating the details of entries maintained by a transaction manager for implementing ~~roll-back~~ rollback procedures in one embodiment.

Please replace paragraph [0014] with the following amended paragraph:

[0014] FIG. 6 is a flow-chart illustrating the manner in which a transaction manager implements ~~roll-back~~ rollback procedures in one embodiment.

Please replace paragraph [0021] with the following amended paragraph:

[0021] FIG. 1 contains pseudo-code illustrating the manner in which an example atomic transaction is implemented in a prior approach. For ease of understanding, atomic transaction Account1( ) (starting at line 105) is shown containing only few task procedures and desired ~~roll-back~~ rollback procedures. However, typical atomic transactions contain many task procedures. Account1( ) is shown containing program logic in lines 110 through 199. Each line is described briefly below.

Please replace paragraph [0023] with the following amended paragraph:

[0023] In line 120, the variable Status is compared with ERROR1 (either a variable set ahead, or a constant value defined elsewhere) to determine whether an error has occurred in the execution of P2( ). Control passes to line 125 if an error has occurred, to line 140 otherwise. Lines 125 (do-reverse-of-P2( )) and 130 (do-reverse-of-P1( )) respectively represent ~~roll-back~~ rollback procedures corresponding to P2( ) and P1( ).

Please replace paragraph [0026] with the following amended paragraph:

[0026] In line 155, a call to task procedure P6( ) is made and the status returned by execution of P6( ) is assigned to a variable Status. In line 160, the variable Status is compared with

ERROR2 (which is similar to ERROR1, described above) to determine whether an error has occurred in the execution of P6( ). If an error has occurred, ~~roll-back~~ rollback procedure corresponding to task procedure P6( ) is executed as indicated in line 165.

Please replace paragraph [0027] with the following amended paragraph:

[0027] In line 170, the content of temp-was-one is checked. If temp-was-one variable is equal to 1, it indicates that control has traversed via task procedures P3( ), P4( ) and P5( ) before reaching P6( ). Accordingly, the ~~roll-back~~ rollback procedures (do-reverse-of-P5( ); do-reverse-of-P4( ); do-reverse-of-P3( )) corresponding to task procedures P5( ), P4( ) and P3( ) are executed as indicated in lines 175, 178 and 180 respectively.

Please replace paragraph [0028] with the following amended paragraph:

[0028] In lines 185 and 190, ~~roll-back~~ rollback procedures corresponding to task procedures P2( ) and P1( ) are executed. In line 195, execution ends in line 195 by virtue of the exit( ) statement. The logic for account1( ) ends in line 199.

Please replace paragraph [0029] with the following amended paragraph:

[0029] One problem with the above approach is that, a program may need to keep track of the control flow (or various lines of code traversed) prior to any situation in which an atomic transaction is to be aborted. The tracked control flow is used to execute the corresponding ~~roll-back~~ rollback procedures. In the illustrative example of FIG. 1, lines 139 and 152 (by using variable temp-was-one) are designed to keep track of whether control traversed lines 145-150, and ~~roll-back~~ rollback procedures of lines 178-183 are executed depending on whether control has traversed lines 145-150. The need to keep track of control adds to the program logic complexity, and may be undesirable particularly complex and lengthy programs.

Please replace paragraph [0030] with the following amended paragraph:

[0030] Another problem with the above-described prior approach is that, code-duplication may be caused due to implementing ~~roll-back~~ rollback procedures corresponding to some task procedures at multiple places (in the entire code). In the above example, ~~roll-back~~ rollback

procedures corresponding to task procedures P1( ) and P2( ) implemented in lines 125 and 130 may need to be duplicated in lines 185 and 190 as well. Such duplication generally leads to unstructured program designs, and could lead to higher implementation and maintenance costs.

Please replace paragraph [0033] with the following amended paragraph:

[0033] Tinit( ) is a procedure which returns a transaction identifier when executed by a program. The transaction identifier may uniquely identify the corresponding instance of the atomic transaction. Tstep( ) is a procedure which enables a user/programmer to specify a task procedure to be executed as a part of an atomic transaction, and a ~~roll-back~~ rollback procedure corresponding the task procedure.

Please replace paragraph [0034] with the following amended paragraph:

[0034] Tabort( ) is a procedure which executes the ~~roll-back~~ rollback procedures specified associated with the procedure tasks executed thus far in relation to an atomic transaction. Tabort( ) may be executed if an atomic transaction is to be aborted. Tabort( ) may accept a transaction identifier corresponding to the atomic transaction as a parameter.

Please replace paragraph [0037] with the following amended paragraph:

[0037] Line 215 is shown containing procedure call Tstep(txid, P1( ), R1( )). The procedure call specifies that P1( ) is a task procedure and R1( ) is the corresponding ~~roll-back~~ rollback procedure. The variable txid specifies that the task procedure P1( ) is to be executed as a part of an atomic transaction uniquely identified by txid. All the Tstep procedure calls of FIG. 2 specify Txid, and the corresponding description is not repeated below in the interest of conciseness.

Please replace paragraph [0039] with the following amended paragraph:

[0039] Line 230 determines if the execution of line 225 returned an error (by checking whether the variable contains a value equaling ERROR1). If an error is encountered (i.e., a situation requiring aborting of the atomic transaction), Tabort ( ) of line 240 is executed to ~~roll-back~~ rollback the procedures executed for the atomic transaction specified by txid.

Please replace paragraph [0041] with the following amended paragraph:

[0041] Line 250 checks whether the variable Temp equals 1. Control pass to line 255 if the condition is true, and to line 270 otherwise. Lines 255, 260, and 265 respectively execute task procedures P3( ), P4( ) and P5( ), with each line specifying the corresponding ~~roll-back~~ rollback procedure.

Please replace paragraph [0042] with the following amended paragraph:

[0042] Line 270 contains a Tstep procedure, which executes task procedure P6( ) having an associated ~~roll-back~~ rollback procedure specified as R6( ). The result of execution of P6( ) is assigned to Status (a variable). In line 275, the content of Status is checked and on occurrence of error (by comparison with ERROR2), control passes to step 280, to step 295 otherwise.

Please replace paragraph [0043] with the following amended paragraph:

[0043] Line 280 contains procedure call Tabort( ) indicating that ~~roll-back~~ rollback is to performed for all the task procedures executed thus far associated with the atomic transaction.

Please replace paragraph [0050] with the following amended paragraph:

[0050] Transaction management block 330 coordinates various operations that may need to be performed in implementing atomic transactions according to various aspects of the present invention. For example, transaction management block 330 may store in memory 350 a transaction identifier generated (by ID allocation block 340) for each atomic transaction. In addition, the ~~roll-back~~ rollback procedures specified with the executed Tstep( ) procedure call may also be stored in memory.

Please replace paragraph [0051] with the following amended paragraph:

[0051] Transaction management block 330 may interface with execution interface 360execute the ~~roll-back~~ rollback procedures stored (in memory 350) associated with an atomic transaction in response to a Tabort( ) procedure call. In addition, transaction management

block 330 may remove from memory 350 all the entries corresponding to an atomic transaction if Tcommit ( ) or exit( ) procedure calls are executed.

Please replace paragraph [0053] with the following amended paragraph:

[0053] FIG. 4 is a table illustrating the manner in which a transaction manager may store various entries to implement ~~roll-back~~ rollback procedures to be performed in one embodiment. For illustration, entries in stack table 400 are shown corresponding to illustrative example in which the program logic of FIG. 2 is executed twice in parallel, and two instances of the atomic transactions (two atomic transactions, in general) are executed in parallel. The atomic transactions are assumed to have been assigned unique identifiers of 200 and 300 respectively.

Please replace paragraph [0054] with the following amended paragraph:

[0054] Table 400 is shown containing four columns 402, 403, 404 and 405, and nine rows 411-419. Columns 402 through 404 respectively represent transaction ID (txid), task procedure and ~~roll-back~~ rollback procedure specified by a corresponding Tstep( ) procedure call, and column 405 indicates the stack status on execution of the corresponding Tstep( ) procedure call. The seven rows correspond to execution of the corresponding seven Tstep( ) procedure calls. Each row is described below in further detail.

Please replace paragraph [0058] with the following amended paragraph:

[0058] Lines 418 and 419 respectively indicate the status of the stack for the two atomic transactions on corresponding execution of line 270 (containing task procedure P6( ) with corresponding ~~roll-back~~ rollback procedure of R6( )). Thus, R6( ) is shown added to the top of the stack for each of the atomic procedures in lines 418 and 419 respectively.

Please replace paragraph [0059] with the following amended paragraph:

[0059] By observing the stack entries, it may be appreciated that execution/support of Tabort ( ) merely requires executing the ~~roll-back~~ rollback procedures in the stack entry of the corresponding atomic transaction. For example, execution of line 280 of FIG. 2 for the first atomic transaction would require execution of R6( ), R2( ) and R1( ) only, whereas for the

second atomic transaction would require execution of R6( ), R5( ), R4( ), R3( ), R2( ) and R1( ) in that order.

Please replace paragraph [0063] with the following amended paragraph:

[0063] In step 540, the user program specifies a combination of the transaction identifier, a task procedure, and a corresponding ~~roll-back~~ rollback procedure. The task procedure contains the program logic implementing at least a portion of the atomic transaction sought to be implemented, and the ~~roll-back~~ rollback procedure may contain the program logic to undo the actions performed by the task procedure. Even though the example above are shown specifying the combination in the form of a single line of code (procedure call), multiple lines can be used in alternative embodiments.

Please replace paragraph [0064] with the following amended paragraph:

[0064] In step 560, the user program determines whether to abort an atomic transaction. Even though the determination is described as being synchronous, it should be understood that the requirements to abort may occur asynchronously as well (e.g., interrupt caused prior to an expected power down). Control passes to step 570 if it is determined to ~~roll-back~~ rollback, and to step 580 otherwise.

Please replace paragraph [0068] with the following amended paragraph:

[0068] In step 610, a transaction identifier that is unique to each atomic transaction may be generated in response to statements such as Tinit( ) described above. In step 630, a combination of the transaction identifier, task procedure, and a corresponding ~~roll-back~~ rollback procedure specified by a user program, may be received.

Please replace paragraph [0069] with the following amended paragraph:

[0069] In step 640, data associating the ~~roll-back~~ rollback procedure with the atomic transaction may be stored. Approaches such as those described with reference to stack memory 350 may be used to store the data. In step 650, the task procedure specified in step 630 may be executed.

Please replace paragraph [0070] with the following amended paragraph:

[0070] In step 660, a determination is made as to whether to abort the atomic transaction, for example, based on program code implementing the atomic transaction. Control passes to step 670 if the ~~roll-back~~ rollback needs to be performed, and to step 690 otherwise.

Please replace paragraph [0071] with the following amended paragraph:

[0071] In step 670, the ~~roll-back~~ rollback procedures are executed in the reverse order in relation to the order of execution of the respective task procedures. Control then passes to step 699.

Please replace paragraph [0078] with the following amended paragraph:

[0078] Secondary memory 730 may contain hard drive 735, flash memory 736 and removable storage drive 737. Secondary memory 230 may store the data (e.g., transaction identifier, task procedure and ~~roll-back~~ rollback procedure etc) and software instructions which cause computer system 700 to provide several features in accordance with the present invention. Some or all of the data and instructions may be provided on removable storage unit 740, and the data and instructions may be read and provided by removable storage drive 737 to processing unit 710. Floppy drive, magnetic tape drive, CD_ROM drive, DVD Drive, Flash memory, removable memory chip (PCMCIA Card, EPROM) are examples of such removable storage drive 737.

Please replace the abstract with the following rewritten abstract:

An aspect of the present invention simplifies the implementation of custom atomic transactions. A program logic (implementing a custom atomic transaction) may request a unique transaction identifier from a programming environment. The program logic may then specify a task procedure, corresponding ~~roll-back~~ rollback procedures, and the transaction identifier using an interface provided by the programming environment. The programming environment keeps track of the specified ~~roll-back~~ rollback procedures. The information maintained by the programming environment may be used to execute the ~~roll-back~~ rollback procedures if the atomic transaction is to be aborted. As the programming environment keeps

track of the ~~roll-back~~ rollback procedures to be executed, the implementation of atomic transactions may be simplified.